CS762: Graph-Theoretic Algorithms
Lecture 18: SP-graphs
February 15, 2002

Scribe: Matthew Skala

Abstract

We know that every simple 2-terminal SP-graph is a partial 2-tree; if we add a restriction for the graphs to be biconnected, then the converse holds as well. We prove that by giving an algorithm to construct the SP-tree for a biconnected partial 2-tree, then define SP-graphs and show that they are exactly the partial 2-trees.

1 Introduction

In the previous lecture we discussed simple 2-terminal SP-graphs, and their relationship to partial 2-trees. Every 2-tree is a simple 2-terminal SP-graph, and each of those is a partial 2-tree; that suggests that simple 2-terminal SP-graphs are in some way very close to the same thing as partial 2-trees. In this lecture, we make that intuition more precise. We try adding a requirement for graphs to be biconnected, and prove that biconnected partial 2-trees are 2-terminal SP-graphs. The proof, given here only as a sketch, involves taking a tree decomposition of a graph and manipulating it with relabelling rules to create an SP-tree. Next we define the concept of simple SP-graphs (not qualified as 2-terminal) and show that those are exactly the partial 2-trees.

2 Definitions

A *k-tree* is defined recursively: the complete graph on k vertices is a k-tree, and a k-tree with n vertices is formed by taking a clique of size k in a k-tree with $n-1$ vertices, and adding a new vertex adjacent to each vertex in the clique. Note that 1-trees and trees are the same class of graphs. A *partial k-tree* is any spanning subgraph of a k-tree. Obviously, any graph G with n vertices is a partial k-tree for some k at most n; in a previous lecture we discussed some properties of the minimum k for which G is a partial k-tree. In the present lecture, we consider partial 2-trees.

We also consider *(s,t)-SP-graphs*, which are defined recursively as follows: The graph K_2, consisting of two vertices s and t and an edge between them, is an (s,t)-SP-graph; and if G_1 is an (s_1,t_1)-SP-graph and G_2 is an (s_2,t_2)-SP-graph, then H_1 formed by identifying t_1 with s_2 is an (s_1,t_2)-SP-graph, and H_2 formed by identifying s_1 with s_2 and t_1 with t_2 is an (s_1,t_1)-SP-graph. The two possible recursive steps, shown in Figure 1, are called "series" and "parallel" (S and P) combination of graphs, by analogy with the construction of electronic circuits. If a graph G is an (s,t)-SP-graph for some vertices s and t, then it is called a *2-terminal SP-graph*. Note that the recursive definition just given allows 2-terminal SP-graphs to contain multiple edges; in this lecture, we primarily consider *simple 2-terminal SP-graphs*, those containing no multiple edges.

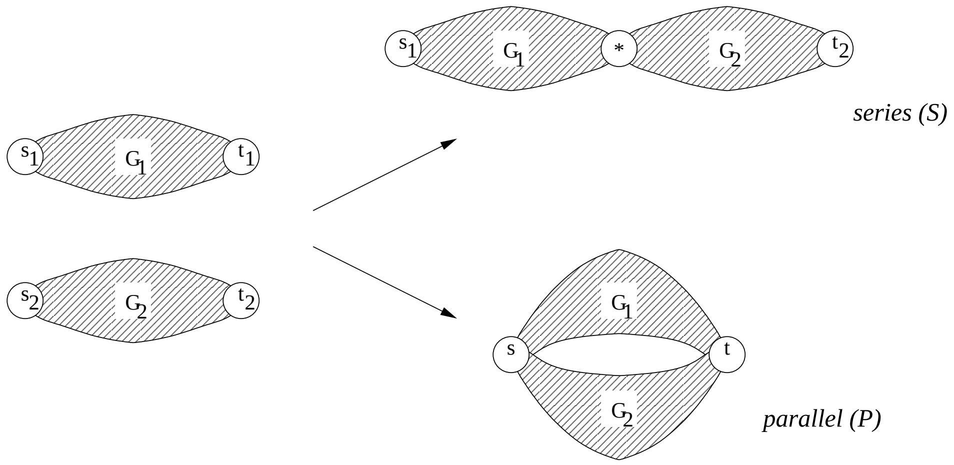

Figure 1: The two combining operations used to define 2-terminal SP-graphs.

The process of building a 2-terminal SP-graph under the recursive definition can be summarized in a structure called an *SP-tree*, which we describe only in general terms here. If G is a 2-terminal SP-graph, then the SP-tree for G has leaves corresponding to the edges of G, and interior nodes of two types called S and P. The S nodes correspond to the series combining operation—each has two child subtrees which are the SP-trees of the two subgraphs combined. Similarly, the P nodes correspond to the parallel operation.

Partial 2-trees can similarly be represented by a tree structure, in this case called a *tree decomposition* of width two. Tree decompositions were discussed extensively in previous lectures; we recall that a tree decomposition for a graph G is a tree where each node has a label consisting of a set of vertices from G. For every edge in G there is some node whose label contains both endpoints. For any vertex v in G, the subgraph of the tree decomposition induced by nodes whose labels contain v is itself a tree. Saying that the decomposition has width two means that each label contains at most three vertices. Note that we generally use the word "nodes" to describe the vertices of the tree decomposition, and reserve the word "vertices" for vertices in the underlying graph, in order to help prevent confusion between the two.

One other concept that becomes important in this lecture is that of biconnectivity. A graph is *biconnected* if there are no vertices that can be removed to leave the graph disconnected. Any such vertices, which prevent the graph from being biconnected, would be called *cut-vertices*. Even if a graph is not biconnected, it can be divided into maximal subgraphs that are biconnected; those are called *biconnected components*. Biconnectivity, also, was a subject for previous lectures.

3 Adding a biconnectivity requirement

We already know that every 2-tree is a simple 2-terminal SP-graph, and that every simple 2-terminal SP-graph is a partial 2-tree.

$$\begin{aligned} \text{2-trees} &\subseteq \text{simple 2-terminal } SP\text{-graphs} && (1) \\ \text{simple 2-terminal } SP\text{-graphs} &\subseteq \text{partial 2-trees} && (2) \end{aligned}$$

Let us consider the converse statements:

$$\text{2-trees} \supseteq \text{simple 2-terminal } SP\text{-graphs} \tag{3}$$
$$\text{simple 2-terminal } SP\text{-graphs} \supseteq \text{partial 2-trees} \tag{4}$$

Both those statements are false; counterexamples are shown in the previous lecture. In this lecture we add another requirement, for all the graphs to be biconnected, and explore how that affects statements 3 and 4. First of all, the new requirement makes no difference to 2-trees.

Theorem 1 *Every 2-tree is biconnected.*

Proof: Recall that a 2-tree G is a chordal graph where in any perfect elimination order, all vertices except the first two have indegree 2. Suppose that G is not biconnected. Consider the vertices in the order of a perfect elimination order of G. One or more vertices at the beginning of the order will be in the same biconnected component, but since the graph is not biconnected, no biconnected component can contain all vertices and so we must eventually encounter some vertex v that is not in the first biconnected component. The indegree of that vertex must be two, and its predecessors form a clique. So v and its two predecessors are a clique, therefore there is exactly one biconnected component containing them all. So v is at least the third vertex of that biconnected component to occur in the perfect elimination order, but that contradicts our choice of v as the second vertex in the second biconnected component. $\square$

Now we can say that every 2-tree is a *biconnected* 2-terminal SP-graph. The 4-vertex cycle is a biconnected 2-terminal SP-graph that is not a 2-tree. So statement 3 is false even when restricted to biconnected graphs. But the 4-vertex star graph, the counterexample to statement 4 from last lecture, is not biconnected. In fact with the restriction to biconnected graphs, statement 4 becomes true: every biconnected partial 2-tree is a 2-terminal SP-graph. That is the subject of the next section, and the main result for this lecture.

4 Every biconnected partial 2-tree is a 2-terminal SP-graph

Now we proceed to the main result for this lecture. Proving this requires many little details, and the resulting induction would be complicated to state rigorously, so we will only give a sketch of the proof.

Theorem 2 *Every biconnected partial 2-tree is a 2-terminal SP-graph.*

Recall the proof of this statement in the other direction from the last lecture. We showed that every 2-terminal SP-graph is a partial 2-tree by associating a special tree, the SP-tree, with the graph. The SP-tree reflected the series and parallel operations used to build the graph. We were able to manipulate it to create a tree decomposition of the graph, which then showed that we were dealing with a 2-tree. Now, we follow essentially the same process in the other direction, to form an SP-tree from a tree decomposition.

Let us take a tree decomposition of a biconnected partial 2-tree G—in particular, a tree decomposition of width 2. We know that such a tree decomposition must exist because G is a partial 2-tree. The tree decomposition forms a model or starting point for our SP-tree. We use the technique proven in a previous lecture to manipulate this tree decomposition until every node has three vertices of G in its label, and the intersection of any two adjacent labels contains two vertices.

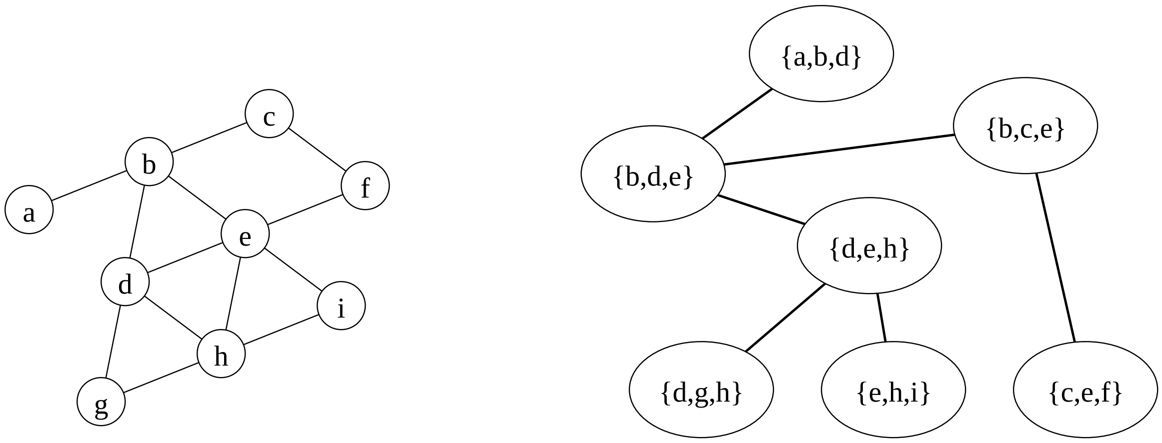

Figure 2: A partial 2-tree and corresponding tree decomposition.

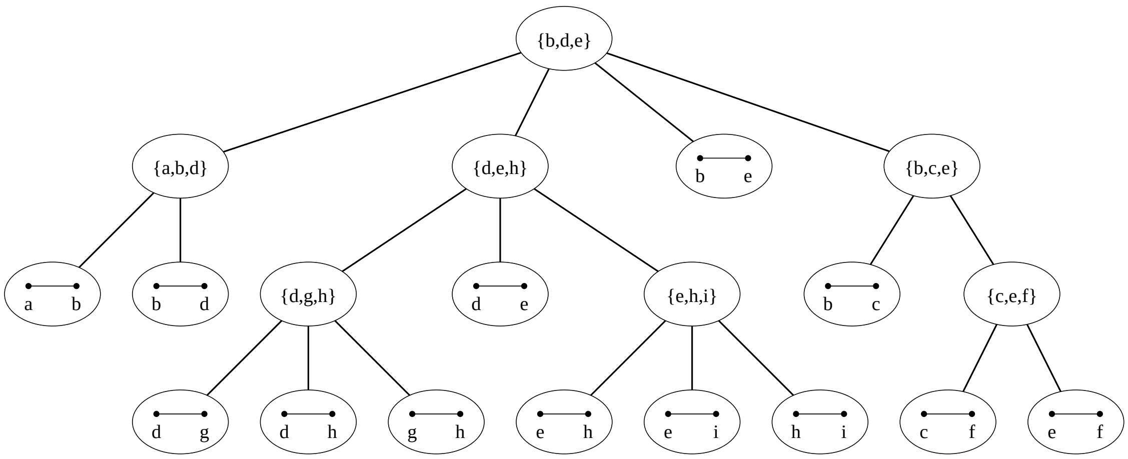

Figure 3: The tree decomposition of Figure 2, rooted and with leaves added for the edges of G.

We will immediately break those conditions, but the purpose is to start with a well-behaved tree decomposition. An example partial 2-tree and such a tree decomposition of it are shown in Figure 2.

For every edge (v, w) in G, we add a tree node containing the edge as its label. We connect the new node to any one of the original nodes that contained both v and w. After adding these new nodes, we root the tree at any arbitrary interior node. Then we will apply rules recursively to change the labels until the tree is an SP-tree.

The tree for our example graph is shown in Figure 3. Note that all the new nodes, labelled with edges, are leaves and end up at the bottom. They look sort of like the leaves of an SP-tree, but the rest of the nodes do not; we fix those starting at the root. As a first step, we replace the label on the root (which should consist of three vertices, v, w, x) with a new kind of label: "terminals: v, w, extra: x". Observe that since we can choose any interior node to be the root, and any of the three vertices of G in its label can be chosen as x, we have a lot of flexibility for which two vertices will be v and w. Those two vertices will eventually be the terminals of the 2-terminal SP-graph; so this observation indicates that there are many possible choices for which two vertices we call the terminals.

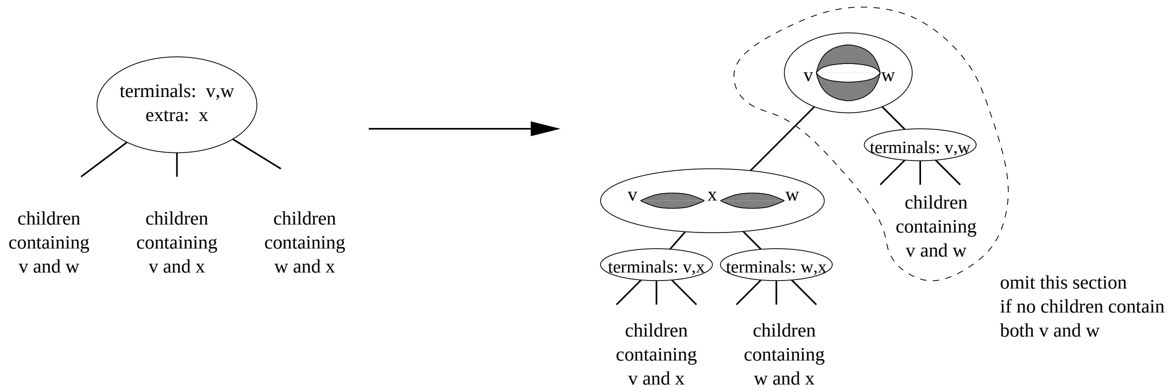

Figure 4: Rule for a node with two terminals and an extra vertex.

That new label is still not an acceptable node label for an SP-graph, so we introduce a recursive rule for replacing such labels. Whenever we consider a label of the form "terminals: v, w, extra: x", we will examine the children of that node. Because of the way we constructed our tree from a tree decomposition with adjacent labels intersecting in a set of two vertices, each child must contains exactly two of the three vertices $\{v, w, x\}$ (counting a child labelled with an edge as containing the endpoints of that edge). There are potentially three kinds of children: those containing v and w; those containing v and x; and those containing w and x. We will replace this node, and re-arrange its children, as shown in Figure 4. If there are no children containing v and w, then we omit the nodes shown in the dotted bubble. Because G is biconnected, v and w may not be cut-vertices, and the construction of the tree then requires that there must be leaves (and so, subtrees) in each of the other two classes of children. That is one of the details that would have to be proven more rigorously in a formal proof.

By applying that recursive rule, we have created a new kind of node label, of the form "terminals: v, w" with no extra vertex. We now need additional rules to deal with those new labels. There are several new rules for replacing a node with terminals and no extra vertex, depending on how many children it has.

If a node is labelled "terminals: v, w", it has one child, and that child is a leaf, labelled with an edge, then we simply remove the interior node as shown in Figure 5. The recursion terminates at that point, because the leaf is acceptable for a SP-tree. If there is one child and that child is not a leaf, then it must be labelled with three vertices $\{v, w, x\}$, the same v and w that are terminals in the label of the current node. Then we replace this node and its child with a node labelled "terminals: v, w, extra: x", as shown in Figure 6, and we can apply the rule already described for relabelling nodes with terminals and extra vertices.

When a node labelled "terminals: v, w" has more than one child, we split it into two nodes with the same label, joined by a parent denoting a parallel combination as shown in Figure 7. The children of the old node are divided between the two new ones, with at least one child on each side. This rule might appear to be counterproductive: we started with one no-extra node we did not know how to label, and ended up with two, without putting any new labels on the children. However, the total number of children of no-extra nodes, not counting the first child of each no-extra node, is reduced by one every time we apply this rule. The process must eventually terminate with every no-extra node having exactly one child, and then we can apply the previous rule to relabel those.

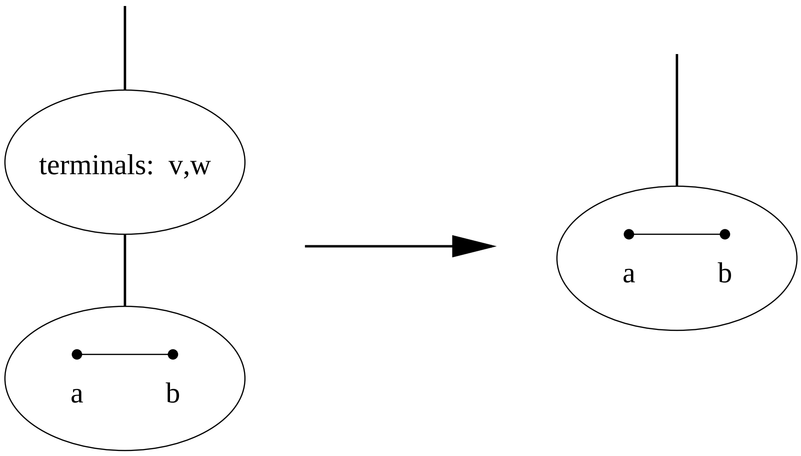

Figure 5: Rule for a node with two terminals, no extra vertex, and one child which is a leaf.

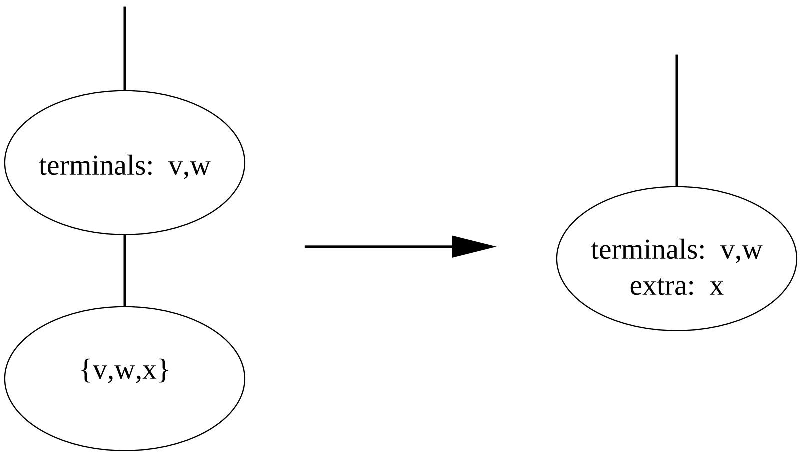

Figure 6: Rule for a node with two terminals, no extra vertex, and one child which is not a leaf.

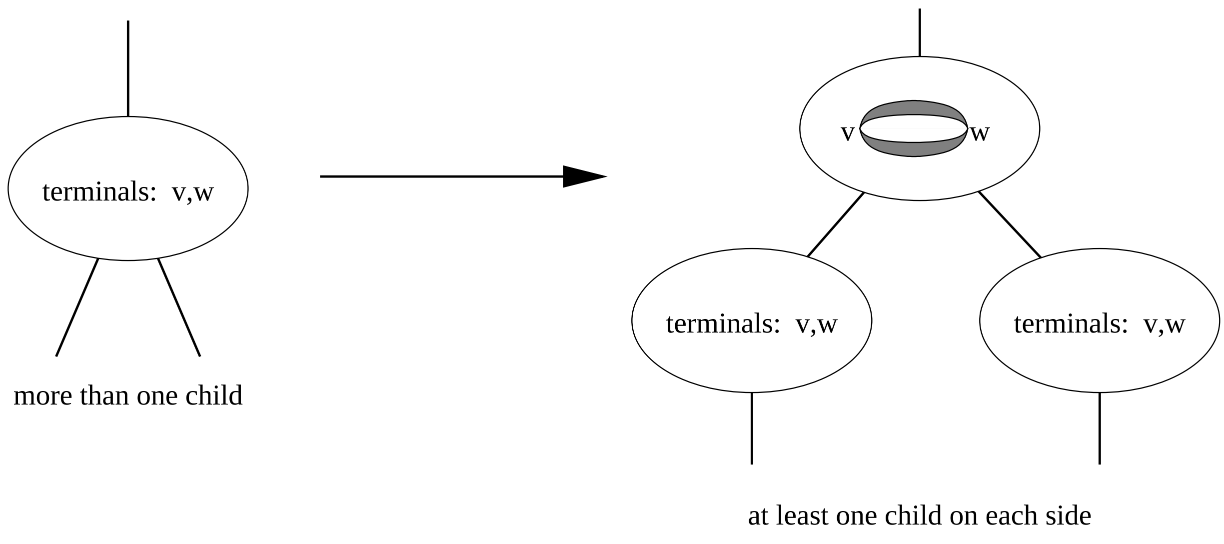

Figure 7: Rule for a node with two terminals, no extra vertex, and more than one child.

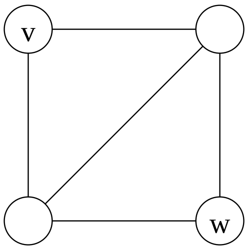

Figure 8: K_4 minus an edge, a simple 2-terminal SP-graph in which the choice of terminals cannot be made arbitrarily.

If we apply the relabelling rules as described until the recursion terminates, we will end up with an SP-tree describing how to build up the graph G in a way that satisfies the recursive definition of 2-terminal SP-graphs; therefore, any biconnected partial 2-tree is a 2-terminal SP-graph.

5 Details and consequences

We need a few more details to tighten up the proof outline of the previous section. First of all, note that we added nodes to what would become the SP-tree for all the edges of G, and we did not remove any nodes while manipulating the tree, so the resulting SP-tree must cover all the edges. Other details we would need to cover to make the proof formal include writing out a precise induction hypothesis, including the properties we are preserving as we do the relabelling.

We have a great deal of freedom about which vertices to choose as the two terminals of the 2-terminal SP-graph, but not total freedom. For the given algorithm to work, the two terminals must both appear in one node of the tree decomposition. In Figure 8, the graph is a 2-terminal SP-graph for some choices of terminals, but there is no way to choose v and w as the terminals. We say without proof that that is because the graph shown plus the edge (v, w) is K_4, the complete graph on four vertices; that the graph is not a (v, w)-SP-graph follows from the following statement, whose proof is left as an exercise.

Claim 1 *Let G be a 2-terminal SP-graph and s and t be two vertices of G. Then G is an (s, t)-SP-graph if and only if $G \cup \{(s, t)\}$ is a biconnected 2-terminal SP-graph.*

Now, why are we doing all this? The results in this lecture give us that biconnected simple 2-terminal SP-graphs are the same class of graphs as biconnected partial 2-trees. Partial 2-trees are the same class as graphs in which every biconnected component is a partial 2-tree, which in turn are the same class as graphs in which every biconnected component is a simple 2-terminal SP-graph. That last class is often of interest, and has its own name.

Definition 1 *A graph G is called an SP-graph or a "confluent" graph if and only if every biconnected component of G is a simple 2-terminal SP-graph.*

Note that we have defined many similar terms here; the definition above refers to "SP-graphs" without any additional qualification, which are not to be confused with 2-terminal SP-graphs, SP-trees, or any of the other concepts. The chain of equivalences can be summarized in the following theorem.

Theorem 3 *Partial 2-trees and simple SP-graphs are the same class of graphs.*

6 Open problems

The proof outline given here is complicated, and a formal proof would be even more complicated. Is there a simpler way to do this? Wald and Colbourn prove in approximately two pages that partial 2-trees are precisely those graphs that do not contain a subgraph homeomorphic to K_4 [WC83], and Bodlaender cites that work as proving that "the class of graphs with tree-width ≤ 2 equals the class of series-parallel graphs." [Bod86] But a really satisfying exposition might also require proving the equivalence of graphs with no homeomorphic K_4, and series-parallel graphs. Technical reports on this subject by Bodlaender and de Fluiter typically prove much more powerful general results instead of the specific problem described in this lecture. [Bod86, BdF95, BdF97]

References

[BdF95] Hans L. Bodlaender and Babette de Fluiter. Reduction algorithms for graphs with small treewidth. Technical Report UU-CS-1995-37, Utrecht University, Utrecht, the Netherlands, 1995.

[BdF97] Hans L. Bodlaender and Babette de Fluiter. Parallel algorithms for series parallel graphs. Technical Report UU-CS-1997-21, Utrecht University, Utrecht, the Netherlands, 1997.

[Bod86] H.L. Bodlaender. Classes of graphs with bounded tree-width. Technical Report RUU-CS-1986-22, Utrecht University, Utrecht, the Netherlands, 1986.

[WC83] J. A. Wald and C. J. Colbourn. Steiner trees, partial 2-trees, and minimum IFI networks. *Networks*, 13:159–167, 1983.